

Nro. ord.	Apellido y nombre	L.U.	#hojas

ORGANIZACIÓN DEL COMPUTADOR I - Parcial

Verano 2016

Ej.1	Ej.2	Ej.3	Ej.4	Ej.5	Nota

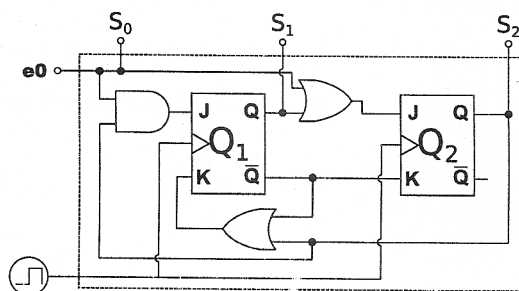
Corrector:

Aclaraciones

- Anote apellido, nombre, LU y numere *todas* las hojas entregadas
- Cada ejercicio se califica con Bien, Regular o Mal. La división de los ejercicios en incisos es meramente orientativa. Los ejercicios se califican globalmente.
- El parcial **NO** es a libro abierto, pero puede tener los apuntes provistos por la cátedra y dos hojas A4 con apuntes propios
- **Importante:** Justifique sus respuestas. Las soluciones a ejercicios de la práctica que se utilicen deben ser incluidas en el examen.
- Son suficientes para aprobar:
 - 4 ejercicios Bien y ninguno Mal.
 - Los ejercicios 1, 2 y 3 Bien y no más de un ejercicio Mal.

Ejercicio 1

- a. Analizar el comportamiento del siguiente circuito y expresarlo en una tabla:



- b. Asumiendo que e_0 siempre vale 0, mostrar la transición de estados que se produce en la memoria interna. Realizar lo mismo asumiendo que vale 1. Para cada una de las dos configuraciones: definir si existen o no estados estables (una configuración es estable si al llegar a ella el próximo valor es sí mismo). En caso de que existan expresarlo, caso contrario justificar por qué.
- c. Extender el circuito anterior con un circuito que tome las salidas del anterior y tenga 3 salidas en la que se expresen:
- Si el numeral $[s_2s_1s_0]$, interpretado en complemento a 2 de 3 bits es el máximo representable
 - Si el numeral $[s_2s_1s_0]$, interpretado en complemento a 2 de 3 bits es el mínimo representable
 - Si la cantidad de bits que valen cero es par

Ejercicio 2 Se tiene una máquina de acumulador –tiene un único registro de propósito general AC– llamada JULIES3. La JULIES3 tiene como unidad direccionable el *nibble* (4 bits) y las palabras son de 12 bits. Cuenta con el siguiente conjunto de instrucciones:

opCode	Instrucción	Descripción
0001	LOAD X	Carga el contenido de la dirección X en AC
0010	STORE X	Guarda el contenido de AC en la dirección X
0011	ADD X	Suma el contenido de la dirección X a AC y guarda el resultado en AC
0100	SUBT X	Resta el contenido de la dirección X a AC y almacena el resultado en AC
0101	INPUT	Lee un valor del teclado y lo guarda en AC
0110	OUTPUT	Escribe el valor de AC en la pantalla
0111	HALT	Termina la ejecución
1000	SKIPCOND C	Saltea la próxima instrucción si se cumple la condición C
1001	JUMP X	Carga el valor X en el PC
1010	XORB	Realiza un XOR entre bits de AC (ver detalle)

Donde X es una dirección de memoria –al ser el único modo de direccionamiento no utiliza la sintaxis [] como su equivalente en la ORGA1– y C una constante de 2 bits (ver descripción a

continuación). La instrucción XORB realiza la operación XOR entre distintos pares de *bits* de AC. El efecto obtenido en el *i*-ésimo *bit* de AC es:

$$AC_i \leftarrow \begin{cases} AC_i \oplus AC_{i-1} & \text{si } i \text{ es impar} \\ AC_{i+1} \oplus AC_i & \text{sino} \end{cases}$$

Las instrucciones son de largo fijo de 12 *bits*, con el siguiente formato:

11	10	9	8	7	6	5	4	3	2	1	0
opCode				Dirección							

Las instrucciones que no utilizan una dirección dejan los *bits* $b_7 \dots b_0$ en cero.

El valor de C (usado en SKIPCOND) se codifica en los *bits* 7 y 6 (los demás *bits* permanecen en cero):

- 00 verifica si $AC < 0$
- 01 verifica si $AC = 0$
- 10 verifica si $AC > 0$
- 11 condición siempre verdadera

Se quiere ensamblar y cargar desde la posición de memoria 0x60 el siguiente código en una JULIES3:

```
main:  LOAD 0xFE
      ADD 0x32
      OUTPUT
      INPUT
      SKIPCOND 10
print: OUTPUT
      SKIPCOND 01
again: JUMP main
end:   HALT
```

- a. Definir a qué posición de memoria corresponde cada etiqueta.
- b. Indicar el contenido de la memoria luego de ensamblar y cargar el código anterior.
- c. Para cada parte de la planilla de seguimiento de la máquina ORGA1, justificar si es necesaria o no dicha celda. Indicar si es necesario agregar nuevas celdas o no para poder realizar el seguimiento.
- d. Modificar la planilla de seguimiento de la ORGA1 y realizar el seguimiento de la JULIES3 con la memoria presentada a continuación hasta decidir si se detiene o no (justificar):

0xA0	0xA3	0xA6	0xA9	0xAC	0xAF	0xB2	0xB5	0xB8	0xBB
0x1B8	0x3B2	0x80	0x9A3	0x2B5	0x8C0	0xDEC	0xAAA	0x914	0x000

El PC inicia en 0xA0, el AC en 0x000 y la memoria toda en cero salvo las posiciones indicadas.

- e. Definir:
 - I. El tamaño máximo de la memoria
 - II. El tamaño del PC
 - III. La cantidad de instrucciones sin operandos que podrían agregarse a este formato de instrucción.
 - IV. Si la cantidad de memoria que tiene una JULIES3 es 64 *bytes*, ¿Afecta esto a las instrucciones? En caso de que no justificar por qué, en caso de que sí dar un ejemplo de dónde es que afecta el normal funcionamiento de la JULIES3.

Ejercicio 3

- a. Realice el diagrama del *datapath* de una microarquitectura para la JULIES3 que soporte la ejecución de las instrucciones descritas (excepto HALT, INPUT y OUTPUT). Recuerde indicar el tamaño de cada registro, de los buses internos y externos, las señales de cada componente y justificar la utilización de cada componente escogido y cada decisión tomada.

Para realizar el *datapath* puede utilizar los siguientes componentes:

- una única ALU que realiza las operaciones suma y resta, sin flags.
- un único incrementador, cuyo única operación es sumar uno.

Al incluirlos detallar cuidadosamente el tamaño de los registros y los nombres de las señales. Cualquier otro componente a utilizar deberá ser implementado e incluido en el examen.

- Escriba las microinstrucciones que debe ejecutar la máquina para realizar el *fetch* de una instrucción (no incluir etapas posteriores del ciclo de instrucción).
- Escriba el microprograma que realiza la parte de ejecución del ciclo de instrucción de las siguientes instrucciones:
 - ADD 0x35
 - SKIPCOND 01
 - XORB

Ejercicio 4 Una computadora ORGA1i mantiene la presión de salida de una máquina de espuma dentro de cierto rango. Existe un sensor de presión de la espuma que tiene mapeado un registro de E/S en la dirección 0xFFFF0. Este registro informa la presión actual (PRESION_STATUS). En R1 se mantiene la cuenta de cuantas veces se verificó la presión desde la última falla. La máquina ejecuta el siguiente programa:

```

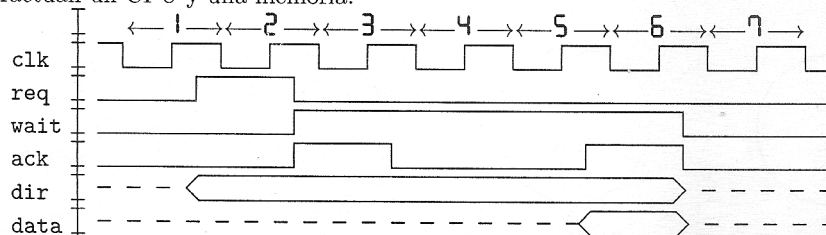
verifPresion:  MOV R0, [0xFFFF0]
                CMP R0, 0x0100
                JG alarmaMuchaPr
                CMP R0, 0x0050
                JL alarmaPocaPr
                ADD R1, 0x0001
                JMP verifPresion
alarmaMuchaPr:  MOV R0, 0xFAFA
                CALL changeP
                JMP verifPresion
alarmaPocaPr:  MOV R0, 0x6868
                CALL changeP
                JMP verifPresion
changeP:        MOV [0xFFFF1], R0
                MOV R1, 0x0000
                RET
  
```

Además se sabe que el ciclo de instrucción de las instrucciones de este programa toma:

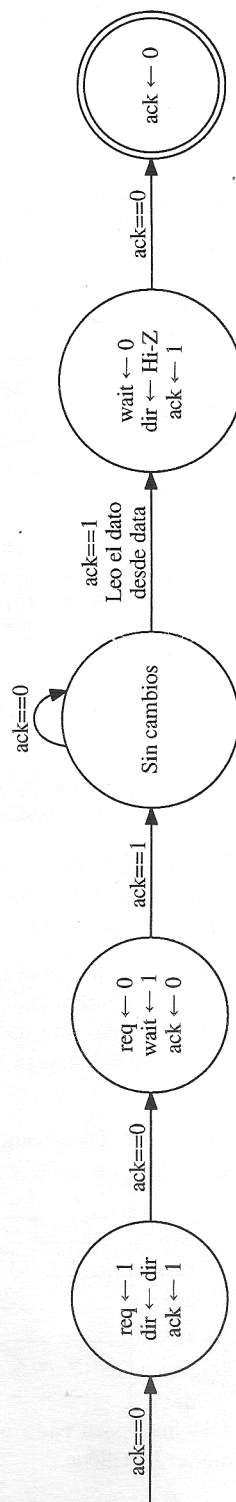
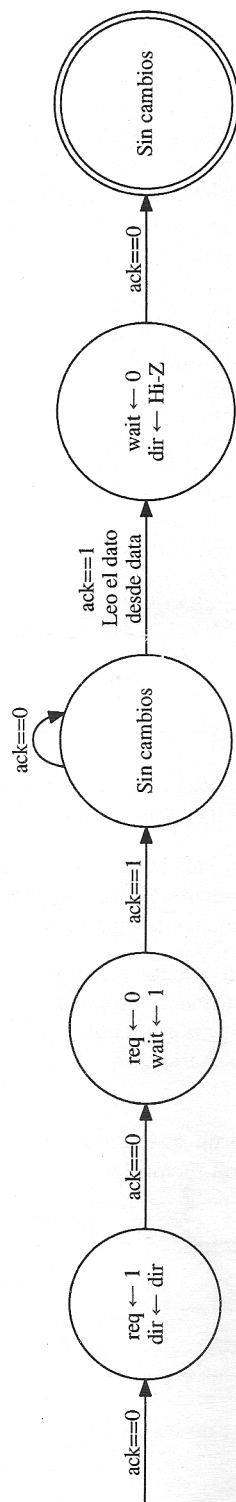
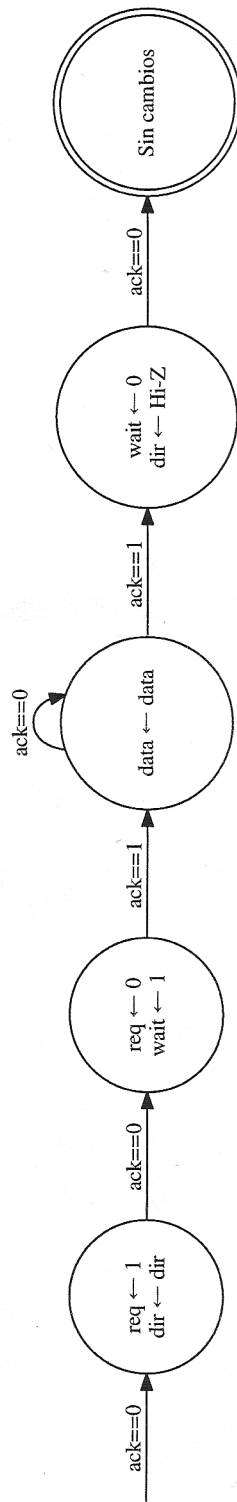
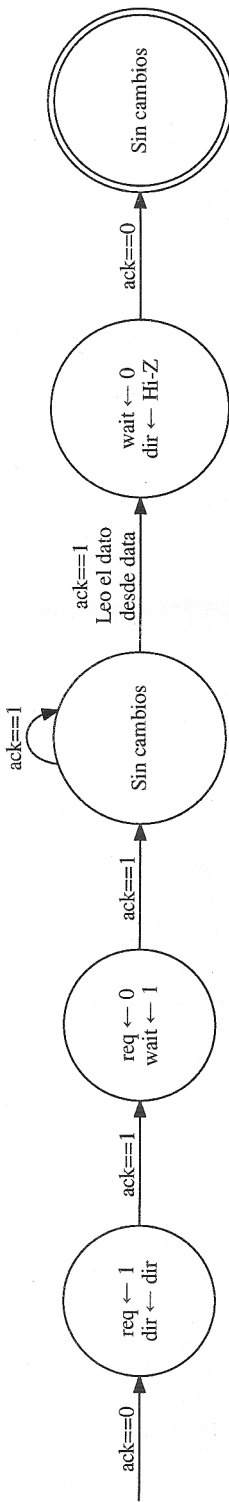
- 3ms para las comparaciones
- 6ms para los movimientos de datos que involucren a la memoria
- 2ms para los saltos incondicionales
- para todas las demás toma 2ms

- Suponiendo que la presión nunca sale de rango, ¿cuántos ms se demora cada ciclo que verifica el estado de la presión? ¿cuál es la frecuencia en Hz del ciclo? (Recuerde que $1\text{ Hz} = \frac{1}{\text{seg}}$)
- Si el sensor de presión solicitara una interrupción cuando la presión supere el valor máximo, modifique el código presentado para aprovechar esta funcionalidad. Recalcular la frecuencia del punto anterior con la nueva configuración.
- Otro modelo de sensor cuenta con una señal que se dispara cuando la presión está fuera de rango. ¿Cuál sería la rutina de atención a esta interrupción? ¿Cómo impacta este nuevo modelo en el código original? ¿En qué contexto conviene tener un dispositivo como éste? ¿Cuál sería la nueva frecuencia de muestreo?

Ejercicio 5 Observando un *bus* se obtiene el siguiente diagrama de tiempos donde se sabe que interactúan un CPU y una memoria:



Definir para cada una de las siguientes máquinas de estado si es compatible o no con el diagrama. Justificar.



7. a.

e_0	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1

Y para todo estado,

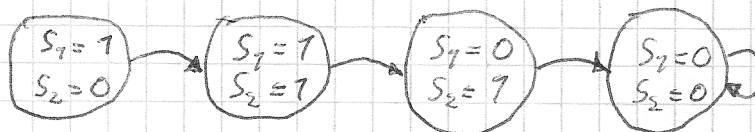
$$S_0 = e_0$$

$$S_1 = Q_1(t)$$

$$S_2 = Q_2(t)$$

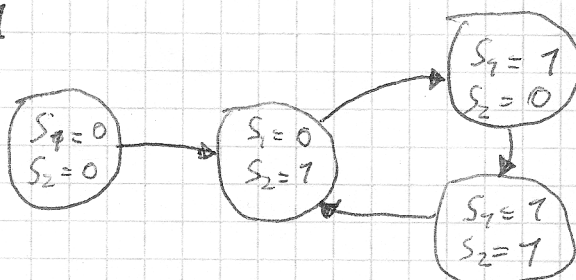
b.

$$e_0 = 0$$

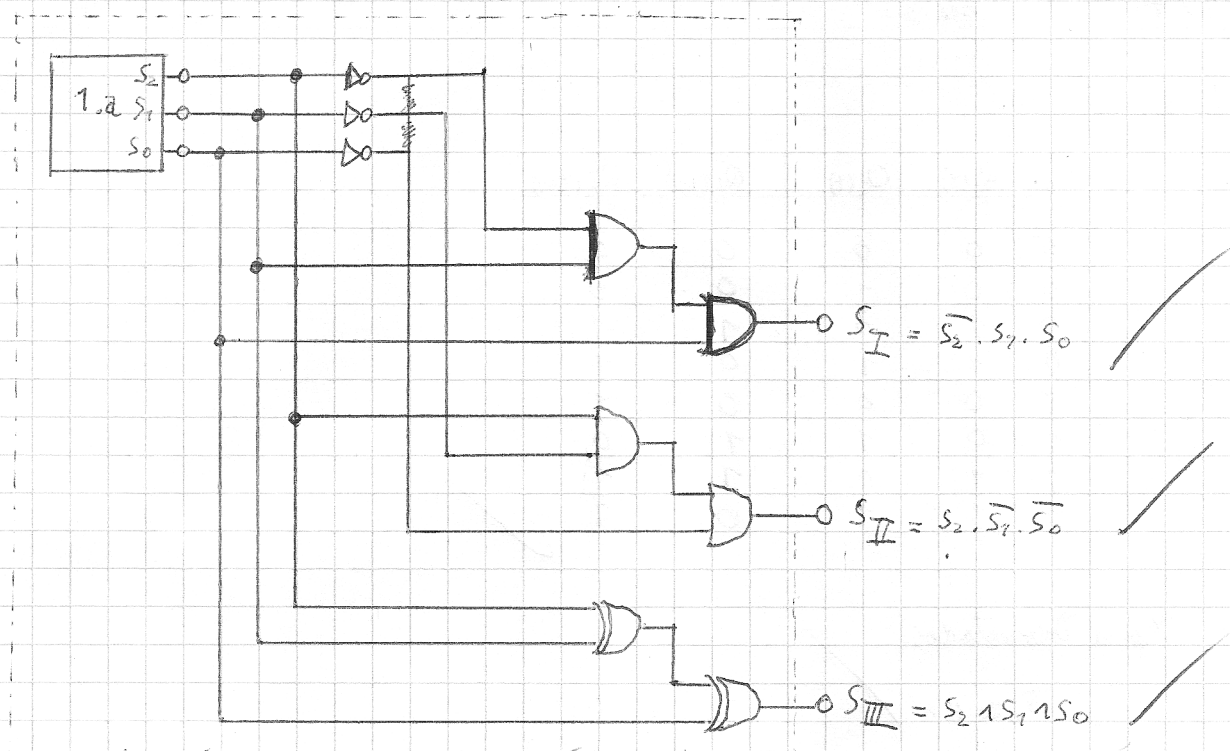
Hay un único estado estable, $S_1=0$ y $S_2=0$ pues

$$\begin{cases} S_1(t)=0 \\ S_2(t)=0 \end{cases} \Rightarrow \begin{cases} S_1(t+1)=0 \\ S_2(t+1)=0 \end{cases}$$

$$e_0 = 1$$



No hay estados estables por ningún estado salta a si mismo (se queda ciclando).

C_n

2. a.

Como hay direccionamiento a ripple y el largo de instrucción es fijo de 12b, cada instrucción va a ocupar 3 posiciones de memoria.

- main está al inicio \Rightarrow main = 0x60

- print está 5 instrucciones después \Rightarrow print = 0x6F

- again " 2 " " \Rightarrow again = 0x75

- end " 1 " " \Rightarrow end = 0x78

b.

ADDR	0x60	0x63	0x66	0x69	0x6C	0x6F	0x72	0x75	0x78
DATA	0x1FE	0x332	0x600	0x500	0x880	0x600	0x840	0x960	0x700

c.

✓ PC - Necesario para saber qué instrucción ejecutar

X SP - No hay stack

X R0, R7 - No tenemos estos registros

✓ Z - Necesario para SKIPCOND (suponiendo que se actualiza cada vez que cambia AC)

X C - No lo utilizamos

X V - No lo utilizamos

✓ N - Idem Z

✓ Memoria - Queremos ver qué ejecutar.

✓ IR - Para ver la instr. actual, pero solo hace falta 1 palabra pues es el largo de todas las instr.

+ AC - El registro que usamos para la mayoría de las operaciones.

e.

I. Direccionamiento a nibble \Rightarrow Celda de memoria = 4b

- Direcciones de 8b $\Rightarrow 2^8$ cantidad de direcciones

\Rightarrow Tamaño máximo de la memoria = $4b \cdot 2^8 = \underline{128 \text{ Bytes}}$ ✓

II Palabras de 12b \Rightarrow PC de 12b X

III Los prefijos $\{0001, 1010\}$ no los podemos usar pues cada operación define que debe haber en los siguientes 8b (cualquier sean ceros).

\Rightarrow Tenemos $\#(\{0000\} \cup \{1011, \dots, 1111\}) = 6$ prefijos de 4b disponibles, cada uno de ellos seguidos de 8b a nuestra disposición.

Si asignamos una instrucción sin operandos a cada combinación de bits tenemos $6 \cdot 2^8 = 1536$ instrucciones posibles.

IV Las instrucciones se ejecutaban normalmente siempre y cuando no accedan a las direcciones de memoria que no están mapeadas a los 64B disponibles.

Si un LOAD, STORE, ADD o SUBT especificaran una dirección inválida o si el PC se fuera de las direcciones mapeadas luego de un JUMP o del ~~fetch~~ fetch, se debería abortar el normal funcionamiento de la máquina. ✓

Planilla de Seguimiento

Valores iniciales:

PC	SP
0x40	X

R0	R1	R2	R3	R4	R5	R6	R7
0x00							

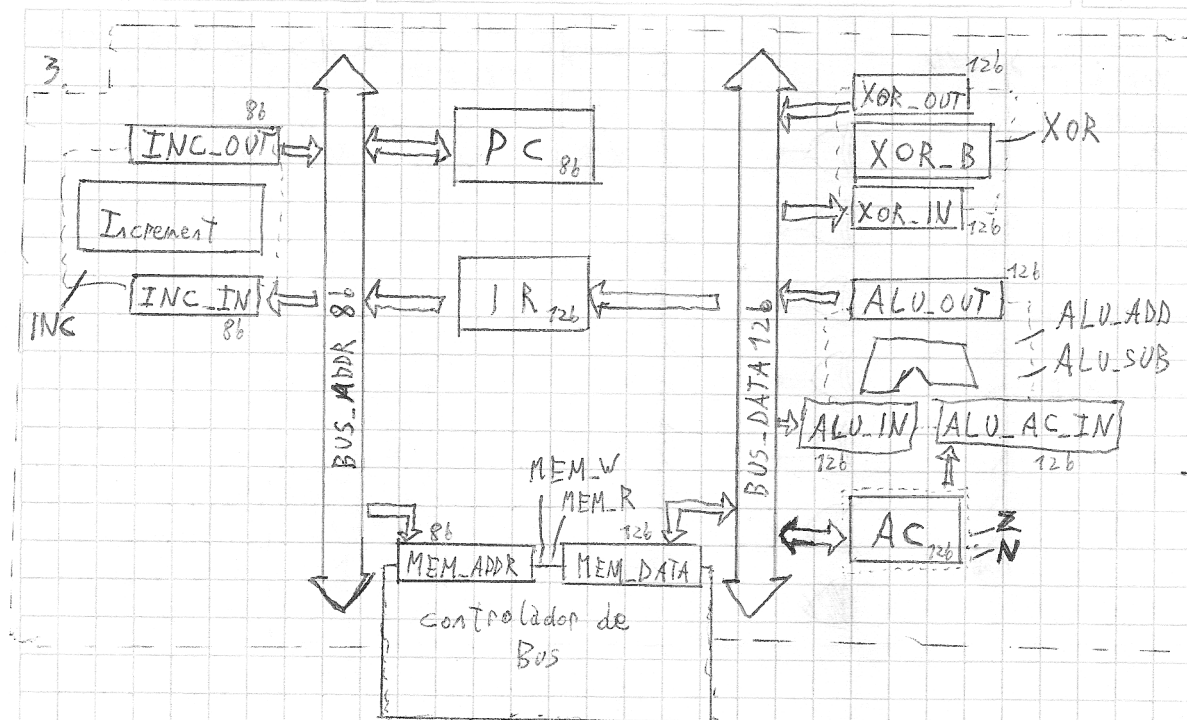
Z	C	V	N
0	X	X	0

Memoria:

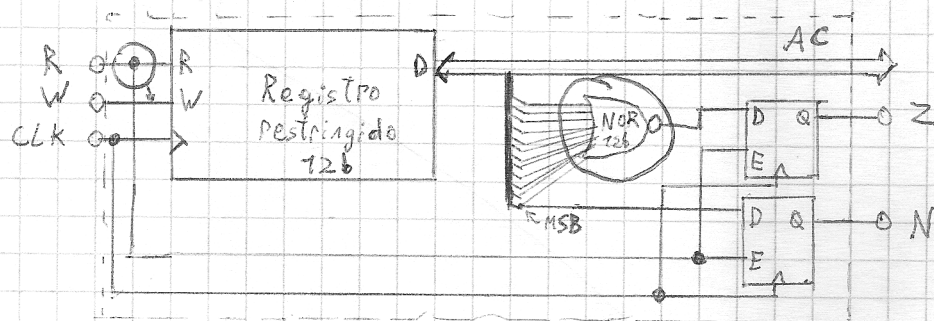
	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0x40	1	B	8	3	B	2	8	8	0	9	A	3	2	B	5	8
0xB0	C	0	D	E	C	A	A	A	9	1	4	0	0	0	0	0

PC	SP	SP+1	IR	Instrucción - 1ª palabra		PC	2da palabra	PC	3ra palabra	PC	Instrucción decodificada	
1	0x40	X	0x1B8	0001 1011 1000		0x43	X	X	X	X	LOAD 0xB8	
Ejecución												Flags ¹ Z C V N
2	0x43	X	0x3B2	0011 1011 0010		0x46	X	X	X	X	ADD 0xB2	
Ejecución												Flags ¹ Z C V N
3	0x46	X	0x880	1000 1000 0000		0x49	X	X	X	X	SKIPCOND 10	
Ejecución												Flags ¹ Z C V N
4	0x4C	X	0x2B5	0010 1011 0101		0x4F	X	X	X	X	STORE 0xB5	
Ejecución												Flags ¹ Z C V N
5	0x4F	X	0x8C0	1000 1100 0000		0x52	X	X	X	X	SKIPCOND 11	
Ejecución												Flags ¹ Z C V N
6	0x55	X	0x700	0111 0000 0000		0x58	X	X	X	X	HALT	
Ejecución												Flags ¹ Z C V N
7		X					X	X	X	X		
Ejecución												Flags ¹ Z C V N
8		X					X	X	X	X		
Ejecución												Flags ¹ Z C V N

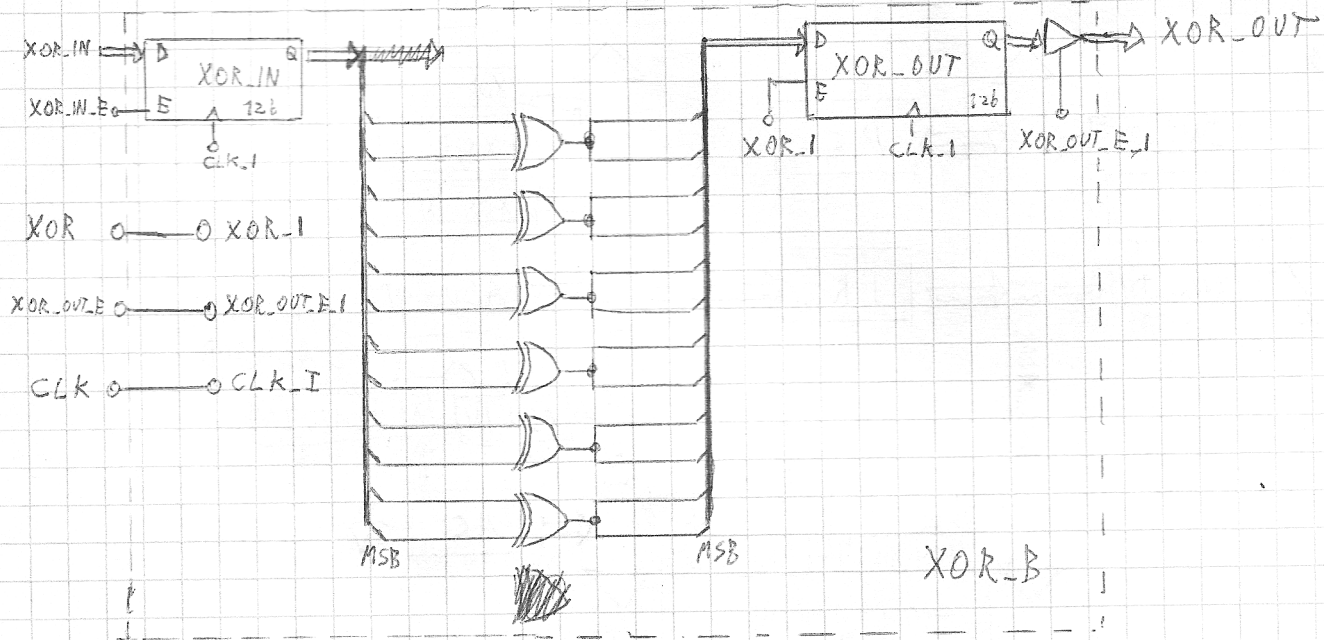
¹ Sólo deben completarse luego de la ejecución de una instrucción que los altere.



- BUS-ADDR maneja direcciones de memoria, por lo que es de 8b
- BUS-DATA maneja palabras de 12b (los datos)
- El PC apunta a direcciones de memoria, de 8b
- El IR aloja instrucciones de 12b y sus 8 bits menos significativos están conectados a BUS-ADDR para usarlos como operandos.
- El incrementador lo usamos para el PC, por lo que trabaja con direcciones de 8b.
- El controlador de bus necesita una dirección de memoria (MEM-ADDR), de 8b y lee o devuelve datos en MEM-DATA del tamaño de una palabra, 12b.
- La ALU trabaja con datos, siempre de 12b.
- El AC es un registro del tamaño de los datos (12b), con el aditamento de dos ~~bits~~ flags de estado como se detalla:



- XOR_B realiza la operación especificada con los datos de 12b ~~26b~~ So circuito interno:



b. Fetch:

MEM_ADDR := PC

MEM_R

~~MEM_R~~

IR := MEM_DATA

INC_IN := PC

INC

→ Aca DEBERÍAS HACER INC_IN := INC_OUT

INC

INC

PC := INC_OUT

5/7

HOJA N°

FECHA

c. ADD 0x35:

MEM_ADDR := IR[0:8] ^[0:7]

MEM_R

ALU_IN := MEM_DATA

ALU_AC_IN := AC

ALU_ADD

AC := ALU_OUT

SKIPCOND 01:

if (Z) {

INC_IN := PC

INC

INC → IDEN b.

INC

PC := INC_OUT

}

XORB:

XOR_IN := AC

XOR

~~XOR~~ AC := XOR_OUT

6/7

HOJA N°

FECHA

4) a)

```

MOV R, C  6ms
CMP R, C  3ms
JG -      2ms
JL > CMP R, C  3ms < 2ms
ADD R, C  2ms
JMP -      2ms

```

~~20ms~~ 20ms

$$t_{\text{ciclo}} = \cancel{20ms} 20ms$$

$$f_{\text{ciclo}} = \frac{1}{20ms} = 50 \text{ Hz}$$

b)

```

verif Presion: MOV R0, [0xFFFF]
                CMP R0, 0x0050
                JL alarmaPocaPr
                ADD R4, 0x0009
                JMP verifPresion

```

```

alarmaPocaPr: MOV R0, 0x6868
                CALL changeP
                JMP verifPresion

```

```

changeP: MOV [0xFFFF], R0
          MOV R1, 0x0000
          RET

```

```

-interrup: MOV R0, 0xFAFA
            CALL changeP
            IRET

```

FALTA SALVAGUARDAR EL VALOR DE R0

$$t_{\text{ciclo}} = 15ms \text{ (una CMP y un salto menos)}$$

$$f_{\text{ciclo}} = \frac{1}{15ms} = 66,6 \text{ Hz}$$

Nota de Interrupción:
 con los ~~distintos~~
 del programa
 original

```

MOV R0, [0xFFFF]
CMP R0, 0x0050
JL alarmaPocaPr
ADD R4, 0x0009
JMP verifPresion

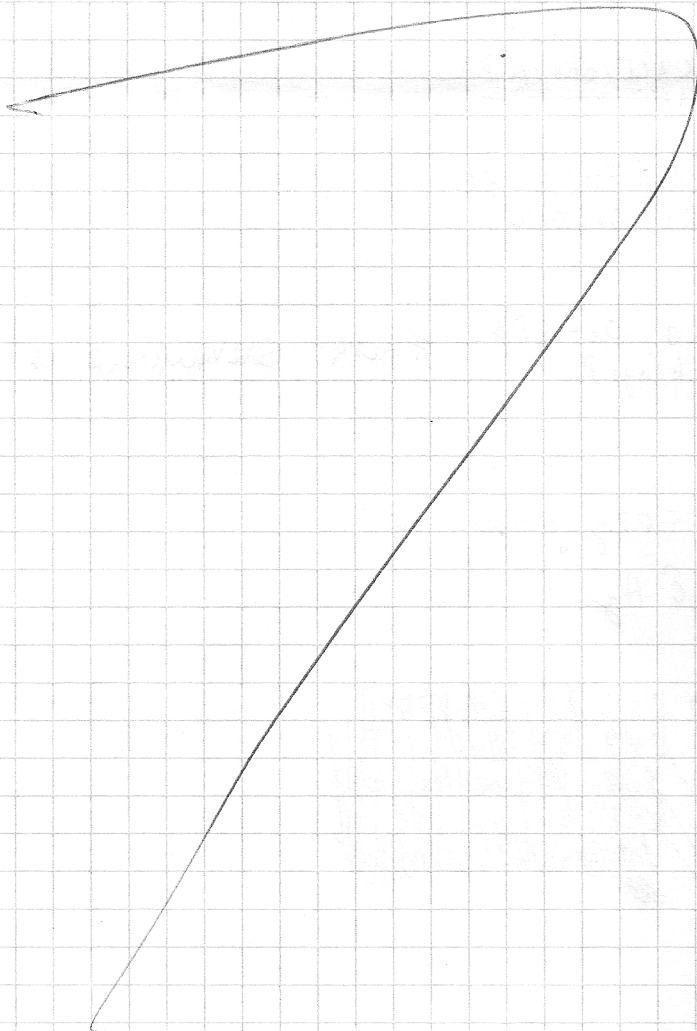
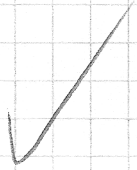
```

c) rutina: MOV R0, [0xFF00]
CMP R0, 0x0100
JG ~~mu~~ muchaPr

MOV [0xFFF1], 0x6868 ; No es mucha presión => es poca presión
IRET

muchaPr: MOV [0xFFF1], 0xFAFA
IRET

- Con este sistema tenemos ~~menos~~ menos retardo hasta que se activa frente a una señal, y liberamos el ciclo principal para que otras cosas que agreguemos corran más seguido. Pero perdemos el contador de muestreos y ocupamos la única ~~señal~~ señal de interrupción, además de tener más delay debido a las comparaciones que en el caso b).



- 5) a. No es compatible, requiere $ack == 1$ en 2-L pero $ack == 0$ en el diagrama. ✓
- b. No es compatible, muestra los datos en data en el 3-H, pero en el diagrama data está en $H1-Z$. ✓
- c. Es compatible, describe perfectamente el diagrama. ✓
- d. No es compatible, usa ack como entrada y salida (no es válido). ✓